Ramin Abbaszadeh Tavassoli

KAGGLE ID: PESKI

Project 2 - Report

CS565

Preprocessing:

The training data had information on 26900 nodes. For each node, we have up to 20 videos that are recommended. The count for the distinct nodes in the training set amounted to 218363. To create the graph, I used the networkx module recommended by Harshal. I first added all nodes. Then I added edges with weights based on the ranking in the training set. The highest recommended node received an incoming edge of weight 1 etc. For some of the 26900 nodes, a bunch of recommendations were missing. For those positions, edges were not added and the weights were skipped. For instance if the 15th recommended node was the only one missing from node j, then edges of weight 1 through 14 would be added from node j to the initial 14 nodes in the order, 15th node would be skipped, followed by edges of weights 16 through 20 added to the rest of the nodes in the order. Doing so, the total edge count amounted to 435587.

Methods:

I started the project after a few sessions of brainstorming with professor Terzi, Kostas, Soham and Anirvan. We exchanged high level ideas on how to approach the problem. We discussed two main approaches, namely graph clustering and shortest path length. Initially we were confident that the highest scores on the leaderboard would be reserved for those who clustered the graph. But this idea proved to be computationally expensive as described in the failed/suboptimal approach section.

I switched gears to implementing the shortest path length. The high level idea was to determine the shortest path length between the source and the target nodes in the test set corresponding to the graph. First all the shortest path lengths were to be computed and stored in a matrix. Then the threshold was to be tuned to optimize score. I shortly encountered two problems. First, the problem was too big and my computer, with python multiprocessing.pool invoked would take 20+ hours to compute all shortest paths. The second disconcerting fact was that 20742 of the nodes that appeared in the test set were nonexistent in the training graph.

The first problem was handled by launching an instance in AWS EC2 with 64 CPU cores. This was the first time I had to use a cloud based computing system and it took a while to get it running but when I eventually got it working at 5 AM a few nights back, The results were astonishing. The problem was solved in just under 2 hours. However, I did not launch the program until I solved the second problem described next.

The nonexistence of nodes in the test data was initially handled by allocating a very high length to any instances of the test data where either the source or the target nodes were absent from the training graph. This ensure that those instances would get $edge\_present = 0$.

I then launched the program in EC2. As mentioned, in just under 2 hours, I got an array of shortest path lengths associated with each source, target instance in the test set. It was then time to find a threshold of acceptance. The optimal threshold was found at 23 where any path length falling below translated to an edge being present. I then revisited the hypothesis I made about the nonexistent test data nodes. I incrementally allowed more and more edges be present when the instance arose by randomly generating from a uniform distribution. Surprisingly, the best score was achieved when all such cases were assigned an edge meaning all cases in the test data where the node data of one (source or target) was missing were connected.

On the last day, I improved my score slightly by employing a Jaccard similarity method. I isolated the 20 recommended videos of each source and for each target, determined the videos that recommended the target. Intersecting the two sets and setting a threshold of 1 (if 1 or more videos intersected between sets, an edge was predicted to exist), I achieved a second score set for the test data. I then cross-referenced the two score sets (shortest path length and Jaccard) and for each decision where Jaccard predicted edge presence and the shortest path did not, I overruled the shortest path decision. This improved the score by 0.02.

Failed/suboptimal Approaches:

Most of my time for this assignment was taken by the idea of graph clustering. Graph clustering is a method involving conductance minimization which algebraically translates into finding the Laplacian and its second eigenvector. Despite storing the 218363 by 218363 Laplacian as a sparse matrix using the Scipy module, finding the second eigenvector was computationally intractable. I ran a fully debugged code for 15+ hours twice overnight and both instances failed to complete. I familiarize myself with the concept of multiprocessing, but soon found out the bitter fact that eigenvector computation is not a dividable process rendering multiprocessing ineffective. This realization led me to focus on the shortest path length alternative.

A suboptimal approach was using a directional graph. Adding directions led to lower scores overall, because it masked the proximity between some pairs. For example, consider a graph of three nodes a, b and c. If $a \rightarrow b$ and $c \rightarrow b$ then there is no path between from a to c even though they are a node apart and both recommend the same video. The undirected graph worked better in capturing relationships.

Improvements:

Three improvements that could possibly make the model better are described in this section. Due to need to study for other exams and partially due to the expensiveness of the EC2 platform ($34 so far), these ideas were not tested but they are worth exploring.

1) Using a directed graph where if we have $i \rightarrow j$ of weight w_{ij} and $j \rightarrow i$ does not exist, add $j \rightarrow i$ with weight w_{ij}. This change puts a path between two proximate nodes the absence of which was substantial shortcoming of the directed graph methodology. Soham implemented this idea and according to him, he managed to raise his score significantly.

2) Weight assignment was done linearly in my model. I would change the function allocation weights to edges to a non-linear function in order to draw the first few recommendations closer and repel the last few more. The function I would have tested first is:

$$
\begin{aligned}
&[int(1.25 ** i) for\ i\ in\ range(1,21)] \\
&= [1,1,1,2,3,3,4,5,7,9,11,14,18,22,28,35,44,55,69,86]
\end{aligned}
$$

3) Using the metadata could result in an improvement. I could potentially leverage the category of the videos such that if the source and the target of the test set shared a genre, it would lower their corresponding shortest path length by a factor. This would make more category-sharing test date to fall below a threshold.
4) If I had more time, I would have liked to polish the Jaccard similarity method to make a better second set of decisions.